Beginner's Guide to C

FIRST EDITION December 2014

Copyright Information

©2014 All rights reserved.

Author Soni

Dedicated to my Parents, my Family, my Lecturers and Teachers

About the Author

Soni a former teacher and is now a technical writer. She holds a degree in commerce and a post graduate degree in computer application. She has hands on experience in programming language and worked as a programmer too. She also has written books on Java that are used by school teachers. She has designed curriculum for online computer courses on programming languages such as Java, C, and C++.

Foreword

This book contains information which will help you to start working with C. Structures, Union and Pointers that are part of C programming language will be covered in the next edition of the book.

You can mail your feedback to gayathrimanoharlal@yahoo.co.in; soni0618@gmail.com

Preface

This book provides information on Programming with C. It has all the information from the basics that will help a beginner to start working with C programming language. As you all are aware that C is the basic programming language that will enhance and build your capability before learning and working with other programming languages. In addition the book gives you simple examples that will help you to work with the programming language.

Contents

Introduction to Programming Languages........8

Introduction to C Language........10

Software programs required for a C program........10

Structure of a C program........12

Compile & Execute C Program........14

Data types........15

The void Type........15

Variables........16

Variable Initialization........17

Keywords........18

Constants........18

Character constants........19

String Constants........19

Defining Constants........19

The const Keyword........20

Escape Sequence........22

Operators........24

Arithmetic Operators........26

Relational Operators........28

Logical Operators........30

The Conditional operator........32

Loops and Conditions........36

Conditional constructs........36

The if construct........37

The if..else construct........38

Switch construct........39

Loop Constructs 43
The while loop 43
The do…while loop..46
The for loop 48
Arrays 52
Character Arrays..54
Library Functions 57
Mathematical Functions 57
String Function 60
The gets and puts Function 60

Introduction to Programming Languages

A computer requires instructions to perform a task. Such instructions are included in a programming language. Collection of such program is software. The procedure of developing software is programming. There are different programming languages. A Computer can be given instructions in any programming language. All computer languages follow a set of rules called syntax. Each programming language uses its own set of syntax.

The 3 main categories in programming languages are, machine language, assembly language and high level language.

Machine Language

Sequence of instructions written in binary form is a Machine Language. Machine languages use 0 and 1 to write instructions. The program written in a machine language execute faster than program written in any other language.

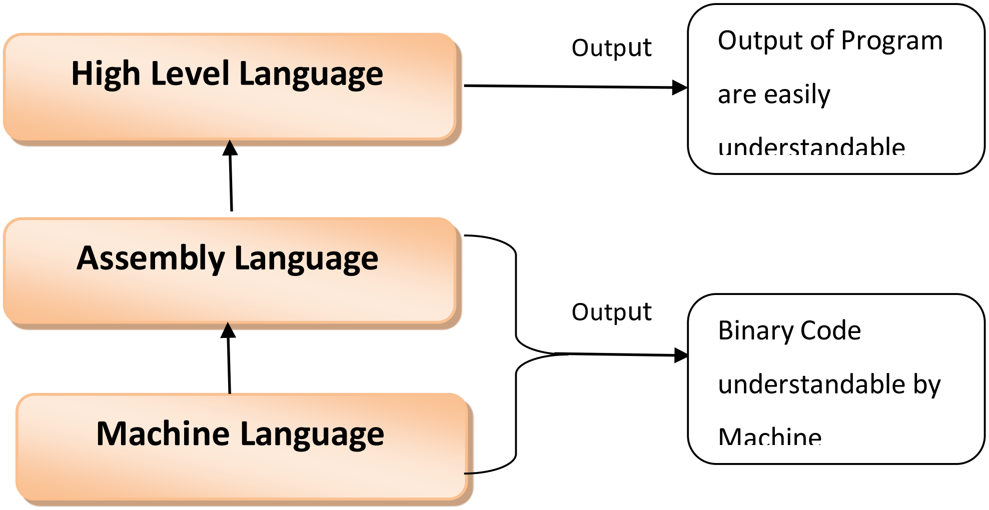
Assembly Language

It is an easy language, as special symbols like special characters, letters, or digits are used to write the program. Computers do not understand the Assembly Language hence the code written in Assembly Language has to be translated to a Machine Language. The Assembler performs the task of translation. After translation of a Assembly Language to a Machine Language the resultant output is an object code in the form of 0 and 1.

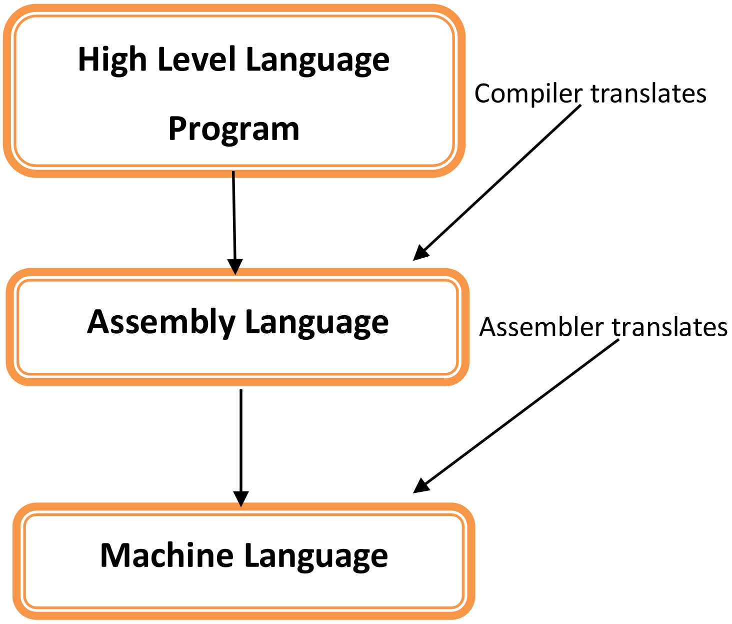
High Level Language

High level languages are easier to understand as English language is used to specify commands. Examples of High Level Language are: C, C++, Java, COBOL, and PASCAL. The High Level Language has to be translated to a machine readable language. A compiler translates the High Level Language. The resultant output is an object code.

The Hierarchy of programming language is:


High Level Language
Assembly Language
Machine Language
Output
Output of Program are easily understandable
Output
Binary Code understandable by Machine

A programming Language conversion to machine understandable language happens the way it is illustrated below:



Introduction to C Language

C is a general purpose, high level programming language and is a widely used language. This language has facilities for structured programming and allows to efficiently provide machine instructions. Hence, the language is used for coding assembly language too. The system software like Unix operating system has been coded using C. C is a procedural language. It was designed to be compiled using a relatively straightforward compiler, to provide low-level access to memory, to provide language constructs that map efficiently to machine instructions, and to have a have need of minimal run-time support. C is useful for many applications that had formerly been coded in assembly language, such as in system programming.

Software programs required for a C program

As no computer can understand C, you need several programs that help the computer understand the instructions. The programs are:

- Editor
- Compiler
- Linker
- Loader

Here Turbo C editor is used for C programming.

The table below explains the purpose of the various programs.

Program	Purpose
Editor	Used to write and alter the content of the program (source code).
Compiler	Used to convert the program to machine language.
Linker	Used to link the header file with the main program before execution.
Loader	Used to load the program to the computer memory for execution.

The basics of a C programming language needs an understanding of the basic building blocks of the programming language, the bare minimum C program contains the following parts:

> *Note*
>
> Compiler is used to check for errors in the program. The complier stores the translated machine language code with .obj extension and is called as object code.

- Preprocessor Commands
- Functions
- Variables
- Statements & Expressions
- Comments

Every C program consists of one or more functions, one of which must be main().

Let us look at a simple code that would print the words "My First Program in C" followed by an explanation of the various parts of the C program:

```
#include<stdio.h>

void main()

{

/* My first program in C */

printf("My Frist Program in C \n");

}
```

Structure of a C program

The various parts of the program given above are explained below:

Preprocessor Commands

1. ***#include<stdio.h>***

   Start writing a C program with a preprocessor command ***#include<stdio.h>.***

   This informs a C compiler to include a header file stdio.h before compiling the program.

   - # is a preprocessor command.
   - Include instructs the C compiler to include the header file library function. The header file contains the instruction for standard input and output.
   - <stdio.h> - All the header file library functions are written within open and close angular brackets, <>.
   - The expansion of stdio.h is “standard input and output header file.
   - .h is the extension of the library function file identified by the compiler with .h extension.
   - All the library functions included in a C program are written as #include<header file name.h>

Refer Library Functions to know about the list of other library functions that can be included in a C program.

2. **main() function**

The next line int main() is the main function where program execution begins.

Note	In C, semicolon (;) indicates the end of statement. #include statement does not end with a semicolon.

All programs in C should have a main() function. There are two variations of main(). They are void main() and int main(). void main() denotes a function does not return a value, int main() denotes a function that is of a numeric return type.

3. **Starting Brace**

All C program start with a left brace {.and the program code always ends with a closing brace - }.

4. **Statements**

Comment Statement

The next line /*...*/ will be ignored by the compiler and it has been put to add additional comments in the program. So such lines are called comments in the program. You can give a brief description about the program within the /*…..*/ comment line statements.

Input and Output statements

Output statement

printf(...)

The next line *printf(...)* is another function available in C which causes the message "My First Program in C" to be displayed on the screen. All the statements that you want to display on the screen should start with the printf( ) function.

Input Statement

scanf("%d", &a);

The data type of the input is specified within double quotes, if it is of integer type then it is % d and &a represents a variable data item.

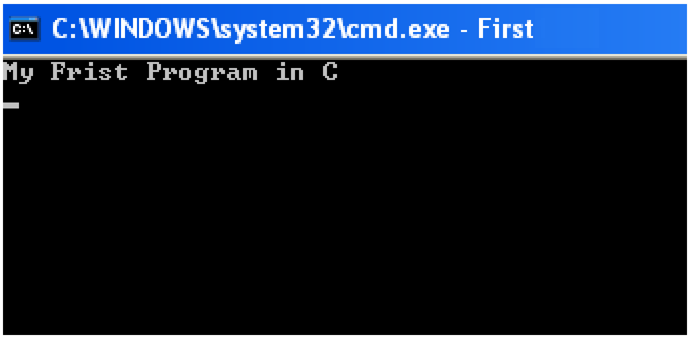
scanf(…..) function can be used to enter any combination of numerical values, single characters and strings.

Compile & Execute C Program

Let’s look at how to save the source code in a file, and how to compile and run it. Following are the simple steps:

1. Open a text editor and add the above-mentioned code.
2. Save the file as First.*c in the TC\Bin directory. The directory where you have the C compiler.*
3. Open a command prompt and go to the directory where you saved the file.
4. Type TC First.*c* and press enter.
5. This opens up the First.c program in the Turbo C window.
6. To compile your code press Alt + F9 key simultaneously.
7. If there are no errors there will be no error message.
8. You have to execute the file to view the output. To execute the file press Ctrl + F9 keys simultaneously.
9. You will be able to see *"My First Program in C"* printed on the screen.

Output screen of First.c



```
My Frist Program in C
_
```

C uses structured programming with variables and recursive statements. In C, all executable code is contained within "functions" All C program use the semicolon as a statement terminator and curly braces({ }) for grouping blocks of statements.

The following are the key points of C language:

- Uses preset keywords
- Includes flow of control structures for, if/else, while, switch, and do/while.
- A large number of arithmetical and logical operators are used.
- Statements are used to specify actions. Common statements are input, output and an expression statement. The expression statement consists of an expression to be evaluated, followed by a semicolon;.

- Functions and procedures may be called and
- Variables may be assigned new values.

The basic characters that are included in a program are:

- Lowercase and uppercase letters: a–z A–Z
- Decimal digits: 0–9
- Graphic characters: ! " # % & ' ( ) * + , - . / : ; < = > ? [\ ] ^ _ { | } ~
- Newline indicates the end of a text line; it need not correspond to an actual single character, although for convenience C treats it as one.

Data types

C supports different types of data, the basic C data types are as listed below:

S.No.	Data Type	Description	Storage Size	Value Range
1.	int	The numeric integer data	2 or 4 bytes	-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647
2.	Char	Single character	1 byte	-128 to 127 or 0 to 255
3.	Float	Floating point number	4 byte	1.2E-38 to 3.4E+38 (Precision up to 6 decimal places)
4.	Double	Double-precision floating point number	8 byte	2.3E-308 to 1.7E+308 (Precision up to 15 decimal places)

The array types and structure types are referred to collectively as the aggregate types. The type of a function specifies the type of the function's return value. We will see basic types in the following section.

The void Type

The void type specifies that no value is available. It is used in the following kinds of situations:

1. When a function does not return a value.
2. For empty set of values.

Variables

Variables store values and they are used to perform calculations and evaluate conditions. A variable refers to values that are not fixed, they keep changing. C programming language also allows defining various other types of variables, other than those defined in the following type:

Variable Type	Description	Example
Integer	A variable, which can have only integers, is called integer variable. Integer variables are used to store whole numbers. A fractional value cannot be stored in an integer variable.	int v=10; int is the data type v is the variable 10 is the value
Character	A variable, which can store only characters, is called character variable. Character constants stored in these variables should be enclosed within single quotes.	char str;; char str= 'A'; where, char is the data type str is the variable , 'A' is value. The char data type values should be provided within single quotes.
Float	Variables, which are used to store decimal values, are called float variables. They can store both integer part (to the left of decimal point) and fractional part (to the right of decimal point).	float marks = 66.78; where, float is the data type marks is the variable 66.78 is the value assigned to the variable.

Variable Definition in C

A variable definition means to tell the compiler where and how much to create the storage for the variable. A variable definition specifies a data type and contains a list of one or more variables of a specific data type.

Variable Initialization

Assigning value to the variable is called Variable Initialization. A variable can be initialized at the place where the variable is declared.

Syntax

data_type variable_name = value;

int x = 15;

float p = 17.1;

char z=’ p’;

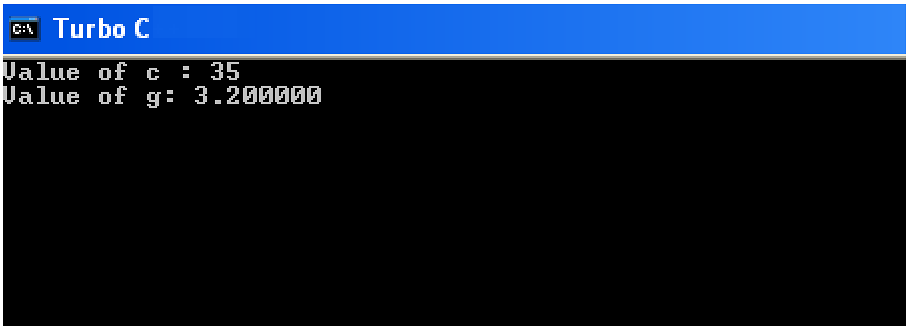
In the following example variables have been declared, but they have been defined and initialized inside the main function:

Example

CODE

```
#include<stdio.h>

int main ()

{

/* variable definition: */
int a, b,c;
float f=3.2, g;
/* actual initialization */

  a =15;

  b =20;

  c = a + b;

  g= f * (b/a);

printf("Value of c : %d \n", c);

printf("Value of g:\ %f", g);

}
```

Result- Output Screen



```
Value of c : 35
Value of g: 3.200000
```

Note	A variable can be initialized anywhere in the program. To view the output as float use %f f type conversion.

Keywords

The keywords are used in program for specifying an instruction, they are reserved words and they are as follows:

auto
break
case
char
const
continue
default
do
double
else
enum
extern
float
for
goto
if
int
long
register
return
short
signed
sizeof
static
struct
switch
typedef
union
unsigned
void
while

Constants

Constants refer to fixed values assigned to a variable in a program. Constants are also called as literals. C has four basic types of constants. They are integer, floating-point, character and string constants. The integer and floating point constants are used to represent numeric values and they are collectively called numeric-type constants.

Character constants

A character constant is a single character and is enclosed in single quotes, e.g., 'x' and can be stored in a simple variable of **char** type.

A character literal can be a plain character (e.g., 'x'), an escape sequence (e.g., '\t'), or a universal character (e.g., '\u02C0').

String Constants

String literals or constants are enclosed in *double quotes "".* A string contains characters that are similar to character literals: plain characters, escape sequences, and universal characters.

You can break a long line into multiple lines using string literals and separate them using whitespaces.

Defining Constants

There are two simple ways in C to define constants:

1. Using **#define** preprocessor.
2. Using **const** keyword.

The #define Preprocessor

Following is the form to use #define preprocessor to define a constant:

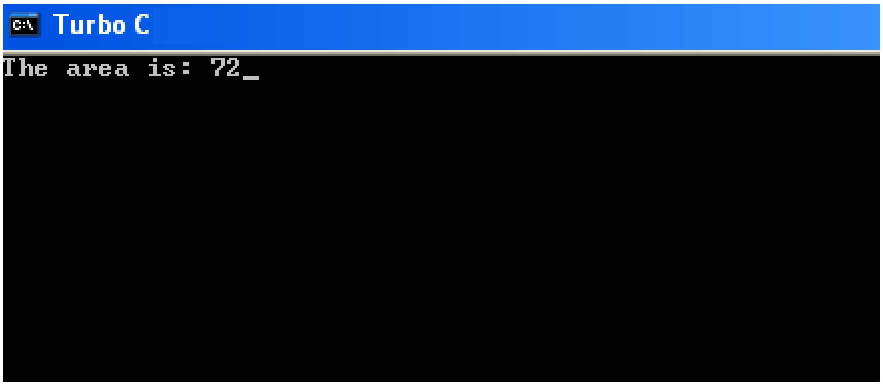
```
#define identifier value
```

Example

CODE

```
#include<stdio.h>

#define LENGTH 12

#define WIDTH  6

int main()

{

int area;

area= LENGTH * WIDTH;

printf("The area is: %d", area);

}
```

RESULT- Output Screen



```
The area is: 72_
```

The const Keyword

You can use **const** prefix to declare constants with a specific type as follows:

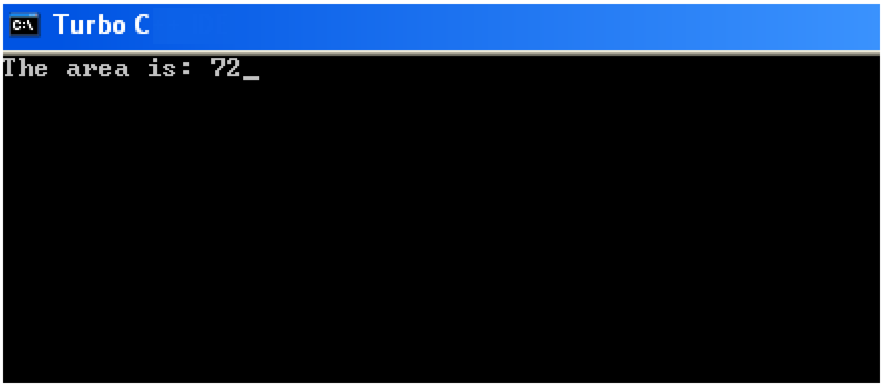
```
const type variable = value;
```

Example

CODE

```
#include<stdio.h>

int main()

{

const int  LENGTH =12;

const int  WIDTH  =6;

int area;

area= LENGTH * WIDTH;

printf("The area is: %d", area);

}
```

RESULT – Output Screen



```
The area is: 72_
```

Note It is a good programming practice to define constants in CAPITALS.

Escape Sequence

There are certain characters in C when they are preceded by a backslash they will have special meaning and they are used to represent like newline (\n) or tab (\t). Given below ere, some of escape sequence codes:

Escape sequence	Meaning
\	\ character
\'	' character
\"	" character
\?	? character
\a	Alert or bell
\b	Backspace
\f	Form feed
\n	Newline
\r	Carriage return
\t	Horizontal tab
\v	Vertical tab

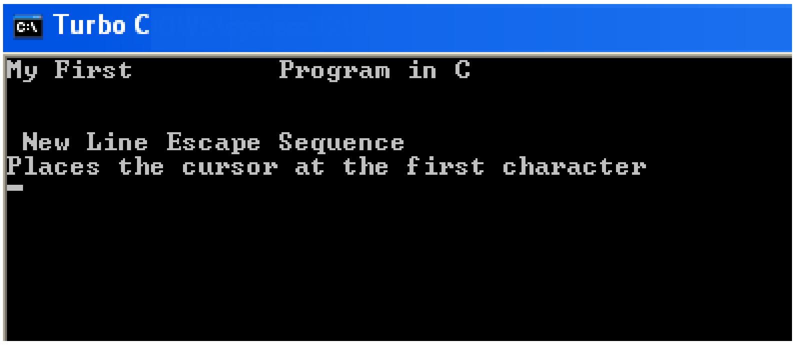
Following is the example to show few escape sequence characters:

Example

CODE

```
#include<stdio.h>
void main()
{
printf("My First\t Program in C\n\n");
printf(“\n New Line Escape Sequence”);
printf(“\n Places the cursor at the first character \r”);
}
```

RESULT – Output Screen



```
My First         Program in C


 New Line Escape Sequence
Places the cursor at the first character
_
```

In the above example the \n , \t and \r escape sequences are used to:

- Display text in a Newline
- Place text with a Tab space
- To position the cursor at the first character

The text is displayed in the next line but the cursor is positioned at the first character. Normally, the cursor is placed at the next line after the text within “” is displayed, but when \r escape sequence is used the cursor is positioned at the first character.

Operators

An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations. Operators and operands together form an expression. C supports set of operators, which are symbols used within an expression to specify the manipulations to be performed while evaluating that expression. C operators are classified into Unary, Binary and Ternary operators.

Operators that use only one operand are termed as unary operators, operators that use two operands to evaluate an expression are termed binary operators and operator that uses three operands are termed ternary operators.

Expression Example

```
a= b + c;
In a = b + c
b and c Operands
+ Operator
```

Unary Operators

The unary operators are used to either increment the value of a variable or decrement it, hence the operators are Increment and Decrement operators.

++ is an Increment Operator

-- is a Decrement Operator

Unary operators use only one operand. The Unary operators can be used as Postfix and Prefix operators.

Postfix Operators

The Postfix operator follows the variable. Example: X++; Y--;

Prefix Operators

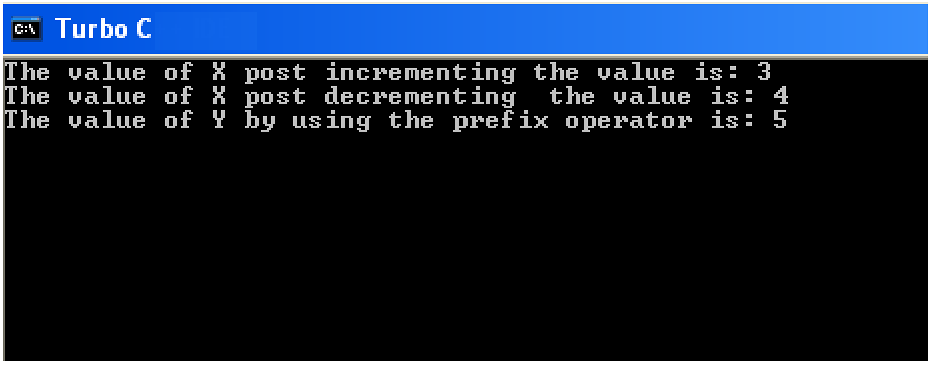
The Prefix operator precedes the variable. Example: ++X; --Y;

Example

CODE

```
#include<stdio.h>
void main()
{
int x=3, y=4;
printf("The value of X post incrementing the value is:
\t%d\n", x++);
printf("The value of X post decrementing  the value is:
\t%d\n", x--);
printf("The value of Y by using the prefix operator is:
\t%d", ++y);}
```

RESULT – Output Screen



```
The value of X post incrementing the value is: 3
The value of X post decrementing  the value is: 4
The value of Y by using the prefix operator is: 5
```

In the above given example code the statement ++y, increments the value of y and then prints the value. But when a post fix operator is used the value of the variable is displayed and then the value is incremented.

C operators are as follows:

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Assignment Operators

Operator Type	Description	Operators
Arithmetic	Arithmetic operators are used to perform mathematical operations like addition, subtraction, multiplication and division.	+, -, *, /, %
Arithmetic Assignment	Assignment operators along with Arithmetic operators allot value to an operand	=, +=, -=, *=, /=, %=, &=, |=, ^=, <<=, >>=
Logical	Logical operators are used to combine two or more expressions.	!, &&, ||
Conditional Operator	The conditional operator is used to evaluate an expression.	?:
Relational operators	Relational operator are used to compare values. It evaluates whether the given expression is TRUE or FALSE.	==, !=, <, >, >=, <=

The following table details the operators with examples.

Arithmetic Operators

Example when A = 10 and B = 20

Operator	Description	Example
+	Adds two operands	A + B will give 30
-	Subtracts second operand from the first	A - B will give 10
*****	Multiplies both operands	A * B will give 200
/	Divides numerator by de-numerator	B / A will give 2

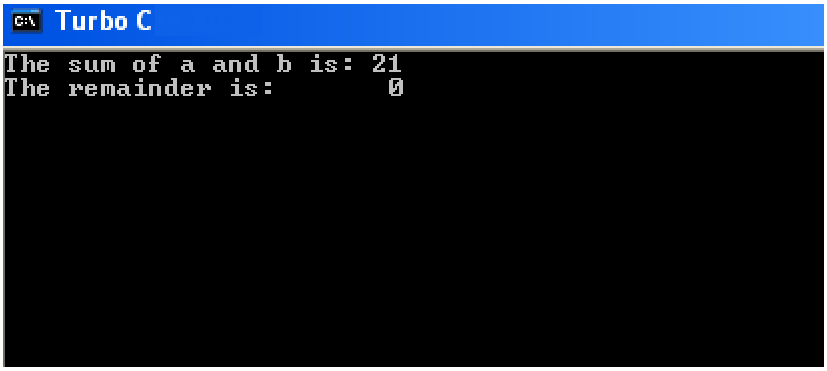
Operator	Description	Example
%	Modulus Operator and remainder of after an integer division	B % A will give 0
++	Increments operator increases integer value by one	A++ will give 11
--	Decrements operator decreases integer value by one	A-- will give 9

Example

CODE

```
#include <stdio.h>
main()
{
int a=18,b=3;
printf(“The sum of a and b is:\t%d\n”, a+b);
printf(“The remainder is:\t%d”, a%b);
}
```

RESULT – Output Screen

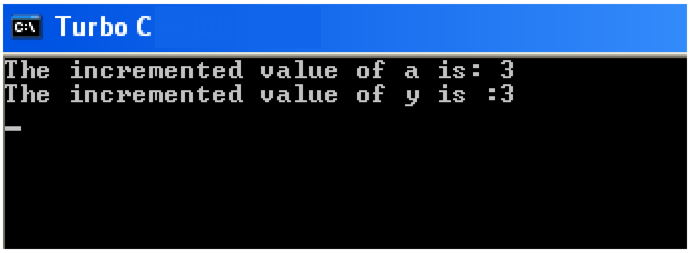


```
The sum of a and b is: 21
The remainder is:       0
```

Code

```
#include <stdio.h>
main()
{
int a=2, y=2;
a++;
++y;
printf(“The incremented value of a is: %d\t\n”, a);
printf(“The incremented value of y is :%d\t\n”, y);
}
```

RESULT – Output Screen



```
The incremented value of a is: 3
The incremented value of y is :3
_
```

Relational Operators

In the following table an example is used to explain all the relational operator, where A =10 and B= 30.

Operator	Description	Example
==	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true, hence the result will be 0.
!=	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true, hence the result of this statement will be 1
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true and the result will be 0.

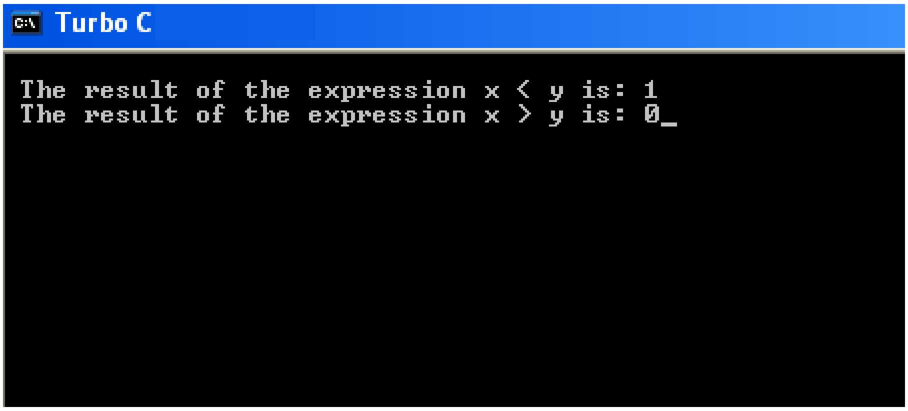
Operator	Description	Example
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true and the result will be 1.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true, hence the result will be 0.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true, hence the result will be 1.

Example

CODE

```
#include <stdio.h>
main()
{
int x = 20;
int y = 65;
int result = 0;
result = x <y;
printf ("\n The result of the expression x < y is: %d”,
result);
result = x >y;
printf ("\n The result of the expression x > y is: %d”,
result);}
```

RESULT – Output Screen



```
The result of the expression x < y is: 1
The result of the expression x > y is: 0_
```

Logical Operators

In the following table an example is used to explain all the logical operator, where A =10 and B= 30.

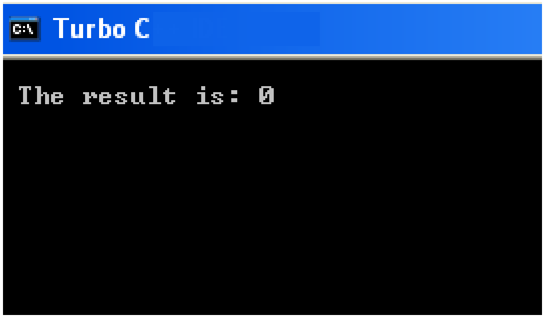
Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non-zero, then condition becomes true.	(A && B) is false.
||	Called Logical OR Operator. If any of the two operands is non-zero, then condition becomes true.	(A || B) is true.
!	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(A && B) is true.

Example

CODE

```
#include <stdio.h>
main()
{
int x = 1010;
int result;
result = (x >1) && (x < 1000);
printf ("\n The result is :%d", result);
}
```

RESULT – Output Screen



```
The result is: 0
```

Arithmetic Assignment Operators

The Arithmetic Assignment operator uses the arithmetic operators (+, -, *, /, %) along with the assignment operator= to form an expression. There are many assignment operators; here a few of them are detailed. In the following table the arithmetic assignment operators are detailed with an example where the values of A, B and C are, 10, 20 and 0.

Operator	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side operand	C = A + B will assign value of A + B into C, hence C=30
+=	Add and assignment operator, It adds right operand to the left operand and assign the result to left operand	C += A is equivalent to C = C + A, hence the resultant value of C will be 0 +10= 10
-=	Subtract and assignment operator, It subtracts right operand from the left operand and assign the result to left operand	C -= A is equivalent to C = C – A, hence the resultant value of C will be 0-10= -10

Operator	Description	Example
***=**	Multiply and assignment operator, It multiplies right operand with the left operand and assign the result to left operand	C *= A is equivalent to C = C * A, hence the resultant value of C will be 0 *10=0
/=	Divide and assignment operator, It divides left operand with the right operand and assign the result to left operand	C /= A is equivalent to C = C / A, hence the resultant value of C will be 0/10=0
%=	Modulus and assignment operator, It takes modulus using two operands and assign the result to left operand	C %= A is equivalent to C = C % A, hence the resultant value of C will be 0%10=0

The Conditional operator

The conditional operator ? : Operator is used to evaluate an expression.

Syntax

variable = (conditional expression) ? TRUE: FALSE

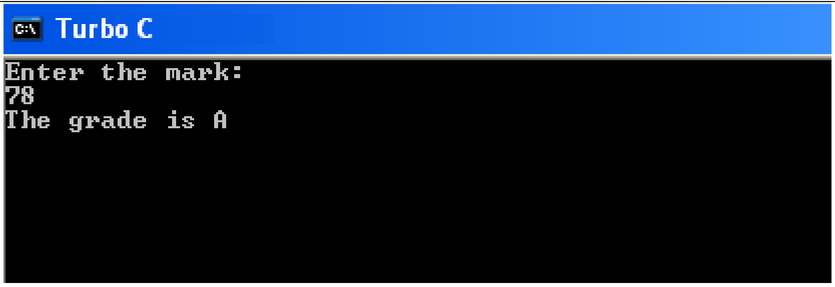
Example

x = (y > z) ? y : z

CODE

```
#include <stdio.h>
main()
{
int mark;
char grade;
printf("Enter the mark: \n");
scanf("%d", &mark);
grade = (mark >= 75)?'A':'B';
printf("The grade is %c\t\n", grade);}
```

RESULT – Output Screen



```
Enter the mark:
78
The grade is A
```

Operators Precedence in C

Operator precedence determines the grouping of terms in an expression. This affects how an expression is evaluated. Certain operators have higher precedence than others; for example, the multiplication operator has higher precedence than the addition operator. For example a = 8 + 3 * 5; here, a is assigned 23, not 55 because operator * has higher precedence than +, so it first the multiplication operation is performed 3 * 5 and then adds with 8.

EXERCISE

1. What will be the output of the program?

```
#include <stdio.h>
main()
{
int a=10, b, c=0;
printf("Enter the first number: \n");
scanf("%d", &a);
printf("Enter the second number: \n");
scanf("%d", &b);
c=a%b;
printf("The resultant value stored in C is:\n%d", c);
}
```

2. Write a program to accept the marks of a student in English, Maths and Science and display them.
3. Write a program to accept a character and an integer. Format the input such that only a maximum of 3 digits is accepted. Print the two values in the same line.
4. Write a program to accept three float values from the user. Format the output such that the values are printed with five digits after the decimal point.
5. Write a program to accept the radius of a circle and print its area and circumference.
6. Write a program to do the following:
    a) Declare four variables n1, n2, r1 and r2.
    b) Accept a number from the user and store it in n1.
    c) Assign n1 to num2.

d) Increment n1 using the postfix increment operator and assign it to r1.

e) Increment n2 using the prefix increment operator and assign it to r2.

f) Print the values of r1 and r2.

g) Compare the values of r1 and r2 using the "equal to" relational operator.

h) Print the result of the comparison.

7. Write a program to accept a number and check if it is divisible by 2 and display the result.
8. Write a program to accept two numbers. Use logical operators and print the output of the comparison.
9. Write a program to accept two numbers. Use relational operators and print the output of the result.
10. Write a program that accepts the salary of an employee .If the salary is greater than 12000, he belongs to grade A, if lesser or equal to 12000, he belongs grade 'B'. Use the ternary operator.

Loops and Conditions

C supports both conditional constructs and loop control statements. A program is usually executed in a sequential order, that is, the order in which it is typed. This is called normal flow of control. In normal flow, the execution starts from the first statement of the program and then goes to the next statement and so on. A condition is incorporated into programs to determine the order of program execution In certain circumstances, there may be need for calling a section of a program to perform repetitive tasks. A Loop construct causes a section of a program to be executed repeatedly for a certain number of times.

A Loop or iteration statement is executed based on certain condition. The repetition continues till the condition is TRUE and when the condition becomes FALSE the loop ends and the control is passed to the statement following the loop. The various loop constructs used in C++ are the while loop, do..while loop and the for loop.

Conditional constructs

Conditional constructs are used to control the flow of the program. Decision making is included in a program for determining the sequence in which the program will execute the instructions.

C programming language provides following types of conditional constructs.

Statement	Description
if statement	An **if statement** consists of a boolean expression followed by one or more statements.
if...else statement	An **if statement** can be followed by an optional **else statement**, which executes when the boolean expression is false.
nested if statements	You can use one **if** or **else if** statement inside another **if** or **else if** statement(s).
switch statement	A **switch** statement allows a variable to be tested for equality against a list of values.
nested switch statements	You can use one **switch** statement inside another **switch** statement(s).

The if construct

The *if* keyword is followed by a logical condition within parentheses. This condition compares the data and a decision is made based on the result of the comparison.

Syntax

```
if(condition)
{
statements;
}
else
{
statements;
}
```

The if..else construct

if .. else construct is a conditional construct used for taking decision, whether a particular action can be performed or not.

Syntax

```
if (condition_1)
    {
       ...block of statements executed if condition_1 is
true
       ...statements are either more ifs and conditions
or normal
          normal lines of code that end with semi-colons
    }
    else if (condition_2)
    {
       ...statements executed if condition_2 is true and
condition_1 was false
    }
    else
    {
       ...statements executed otherwise meaning 1 and 2
were false
    }
```

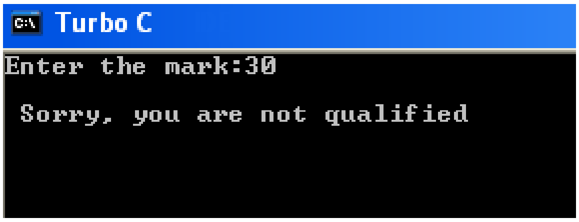
Example

CODE

```
# include<stdio.h>
main()
{
int mark;
printf(”Enter the mark:”);
scanf(“%d”, &mark);
if (mark > 40)
{
```

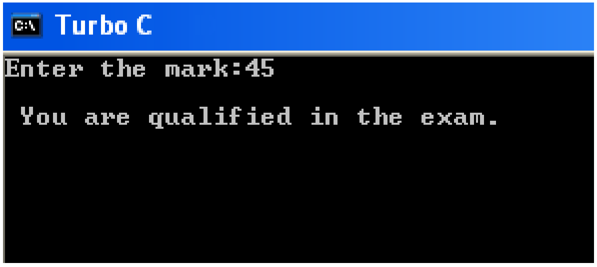
```
printf("\n You are qualified in the exam.");}
else
{ printf("\n Sorry, you are not qualified");
}
}
```

RESULT – Output Screen



```
Enter the mark:30

 Sorry, you are not qualified
```

Using another input the output is as shown below:



```
Enter the mark:45

 You are qualified in the exam.
```

Switch construct

The switch..case construct is a built in multiple branch selection statement. It is used when there can be multiple values for a variable.

Syntax

```
switch (variable)
{
   case const_1:
   {
        ...block of statements...
         break;
   }
   case const_2:
   {
```

```
            ...block of statements...
             break;
        }
        default:
        {
            ...block of statements..
        }
    }
```

When a switch construct is executed, its condition-variable (value of variable) is evaluated and compared with each case constant. If any one of the case constant is equal to the value of the variable then the control passes to the statement following the matched case label. If no case constant matches with the value of the variable, then the control passes to the default label and the statements following default are executed.

A break statement is used to exit case statement. If a break statement is not used; the control will not exit the switch construct. It will continue to execute all the cases, even if a particular case is satisfied.

Example

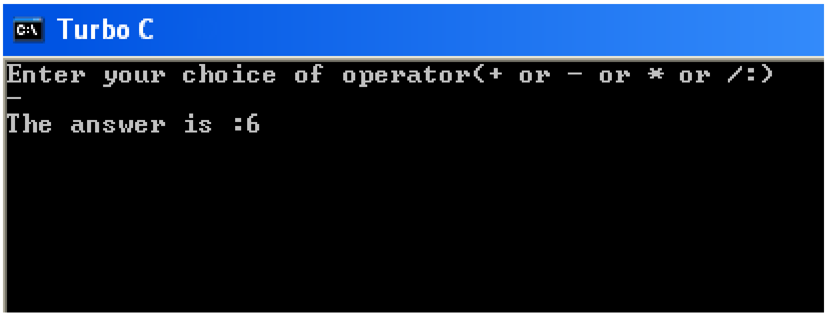
CODE

```
#include <stdio.h>
void main()
{
int ans,num1,num2;
char operator;
num1=8;
num2=2;
clrscr();
printf("Enter your choice of operator(+ or - or * or
/:)\n");
scanf("%c", &operator);
switch(operator)
```

CODE

```
{
case '+':
ans=num1+num2;
printf(" The answer is :%d", ans);
break;
case '-':
ans=num1-num2;
printf(" The answer is :%d", ans);
break;
case '*':
ans=num1*num2;
printf( " The answer is :%d", ans);
break;
case '/':
ans=num1/num2;
printf( "The answer is :%d\t", ans);
getch();
break;
default:
printf( "Wrong choice");
}
getch();
}
```

RESULT – Output Screen



```
Enter your choice of operator(+ or - or * or /:)
-
The answer is :6
```

EXERCISE

1. Write a program to accept dimensions of a square and calculate its area. Both the input and the output should be float values.
2. Write a program to accept numbers till the user enters a zero. Find the sum of these numbers and display the result.
3. Write a program to print the sum of all even numbers within 100.
4. Write a program to accept a number and check if it is less than 100, 200 or 300. Print appropriate messages for each condition. If none of the conditions is satisfied, print the message "Number greater than 300". (Hint: Use the else..if statement).
5. Write a program that accepts a day of the week in number and displays the corresponding day of the week in name using switch...case construct. (Example: If the user enters 1 then display "Monday")
6. Write a program to accept a number and display the number in words. (Example if the entered number is 0, then display the text as Zero). Exit from the program when the user enters number 10.
7. Write a program using switch…case construct to find whether the character given as input is a vowel or not.
8. Write a program to find the greatest of three numbers using Nested if construct.
9. Write a program using the if…else construct to find whether the given number is positive or negative.

Loop Constructs

A Loop construct causes a section of a program to be executed repeatedly for a certain number of times. A Loop or iteration statement is executed based on certain condition. The repetition continues till the condition is TRUE and when the condition becomes FALSE the loop ends and the control is passed to the statement following the loop. The various loop constructs used in C++ are the while loop, do..while loop and the for loop.

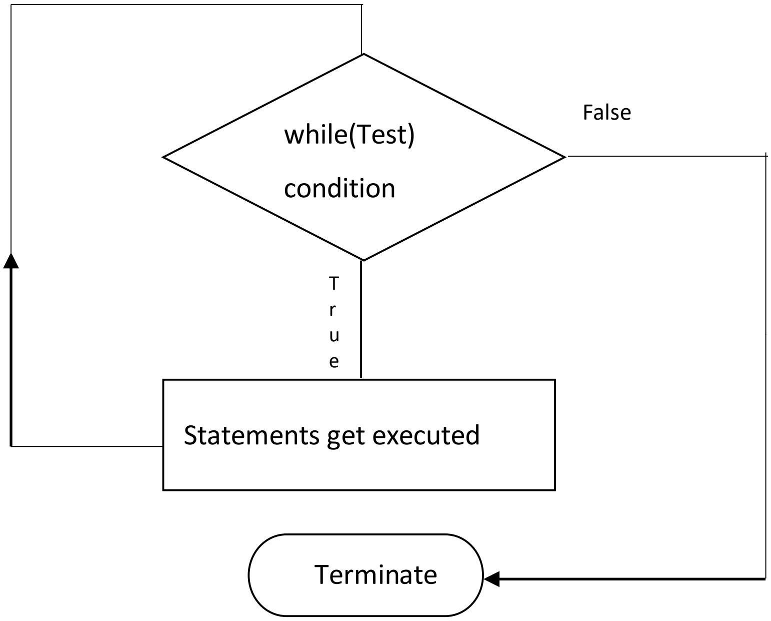
C programming language provides the following types of loop to handle looping requirements.

Loop Type	Description
while loop	Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.
for loop	Execute a sequence of statements multiple times and abbreviates the code that manages the loop variable.
do...while loop	Like a while statement, except that it tests the condition at the end of the loop body

The while loop

Syntax

```
while(condition)
{
statements;
}
```



The while keyword is followed by a condition given within brackets.

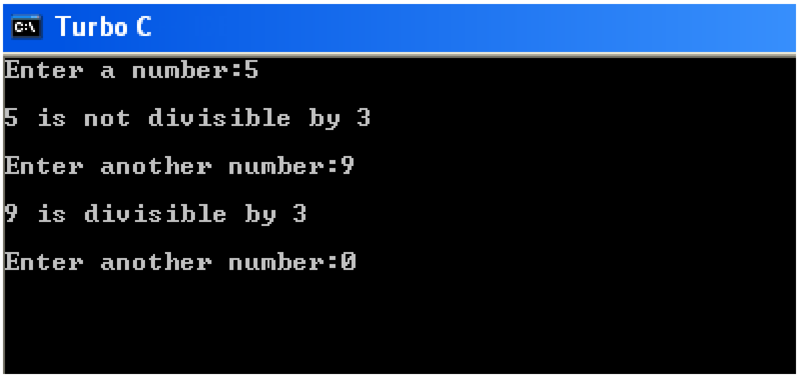
1. The statements (section of program), which should be repeated, are given within curly braces.
2. The condition is first checked before entering the loop.
3. If the condition returns TRUE, the statements inside the loop are executed and the control again returns to the while statement for evaluating the condition.
4. Step 3 is repeated till the condition returns TRUE.
5. If the condition returns FALSE, the loop terminates and the control is passed to the statement following the loop.

CODE

```
#include <stdio.h>
main()
{
int n;
printf("Enter a number:");
scanf("%d", &n);
```

```
while (n!=0)

{

if((n %3)== 0)

printf("\n%d is divisible by 3\n",n);

else

printf("\n%d is not divisible by 3\n", n);

printf("\nEnter another number:");

scanf("%d", &n);

}
```

RESULT – Output Screen



```
Enter a number:5
5 is not divisible by 3
Enter another number:9
9 is divisible by 3
Enter another number:0
```

In the example program given above the program is executed until you enter 0. A loop condition is executed. Inside the loop an if else condition is used to check a condition for divisibility of a number. The number that you enter checks whether it is divisible by 3 and displays the output. As soon as you enter 0 and hit the Enter key the program terminates.

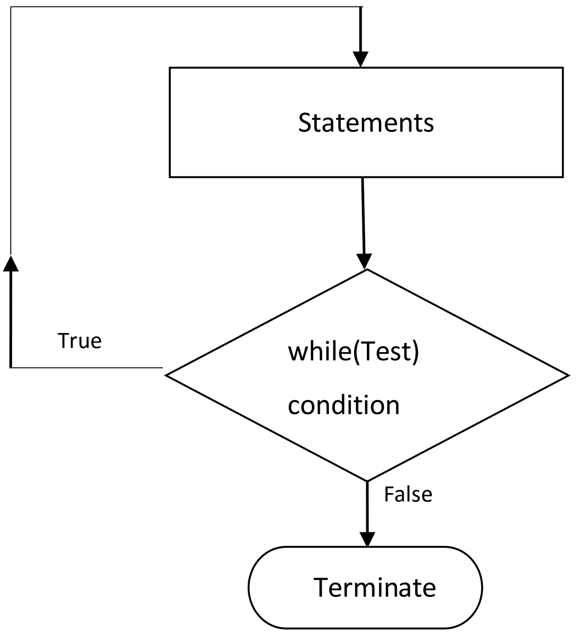
The do…while loop

The do..while loop is used when the body of the loop should be executed at least once without checking the condition. It is an exit-controlled loop. There is a difference between do..while loop and the while loop. In a while loop, the test condition is evaluated at the beginning of the loop. If the test expression is FALSE, the loop body is not executed at all. If the loop body should be executed at least once, then the do… while loop construct is used. In a do…while loop the test condition is given at the end of the loop.

Syntax

```
do
{
statements;
} while (condition );
```

The statements following the do keyword should be given within curly braces. The while condition is given at the end of the loop. If the condition returns TRUE, the loop is repeated and if the condition is FALSE the loop terminates.

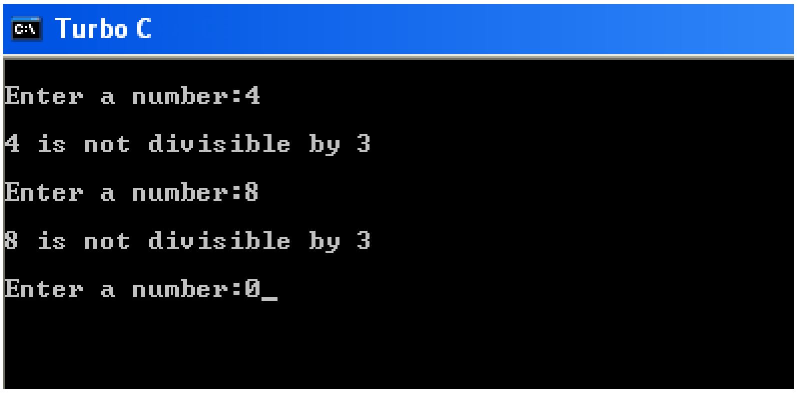


Example

CODE

```
#include <stdio.h>
main()
{
int num;
do
{
printf("\nEnter a number:");
scanf("%d", &num);
if((num%3)== 0)
printf("\n%d is divisible by 3\n",num);
else
printf("\n%d is not divisible by 3\n", num);
}while (num!=0);
}
```

RESULT – Output Screen



```
Enter a number:4
4 is not divisible by 3
Enter a number:8
8 is not divisible by 3
Enter a number:0_
```

In the example program given above, the statements to check the divisibility of a number are given within a do while loop. All the statements are executed until the condition is met. '0' is the condition, when you enter the number as 0 the loop terminates and the program stops execution.

The for loop

The for loop provides a step-by-step procedure for performing an action. The for loop is used when the number of iterations are known .The difference between the for loop and the while loop is that, in the while loop the number of iterations are not known where as in the for loop the number of iterations are known.

Syntax

```
for (initialisation ; test-condition ;
incrementation/decrementation)
{
statements;
}
```

Note	In for and while loops, the loop condition should not end with a semicolon.

The for keyword is followed by initialisation, condition and incrementation or decrementation of the loop separated by a semicolon and enclosed within parentheses. The body of the loop (statements within the loop) is enclosed within curly braces.

Example

```
for ( var=0 ;var<=10 ;var++)
In this,
var=0 - initialisation
var<=10 - condition
var++ - incrementation
var - counter variable
```

Initialisation Expression

The initialisation expression is executed only once, when the control is passed to the loop. It gives the counter variable an initial value.

Test-Condition

The test-condition is executed each time the control passes to the beginning of the loop. The statements given within the body of the loop are executed only after the condition is checked. If the condition is TRUE, the statements within the loop are executed otherwise the control passes to the statements following the loop.

Increment/Decrement Expression

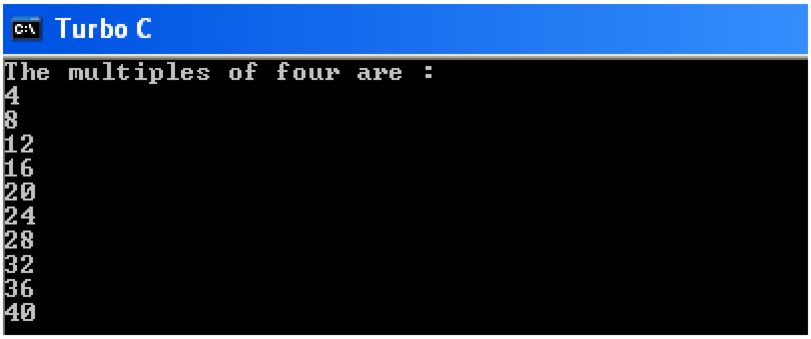
This is executed when the control returns to the beginning of the loop.

Example

CODE

```
#include<stdio.h>
void main()
{
int n,m=0;
clrscr();
printf("The multiples of four are :\n");
for(n=1;n<=10;n++)
{
m=n*4;
printf(“%d \n”, m );
}
getch();
}
```

RESULT – Output Screen



```
The multiples of four are :
4
8
12
16
20
24
28
32
36
40
```

In the above given example program the variable:

- Inside the for loop has been initialized
- A condition when the loop should terminate i.e., <=10 has been specified
- The variable has been incremented
- Inside the for loop the statements are specified.

EXERCISE

1. Write a program to accept 20 numbers using a for loop and display their squares.
2. Write a program, that displays the numbers from 1 to 50 in the descending order using while loop.
3. Write a program, which accepts numbers from the user, finds whether the number is divisible by 6 and displays them using the Do…While loop.
4. Write a program to find the sum of natural numbers from 1 to 20
5. Write a program to accept a number, find the sum of accepted numbers and display them.
6. Predict the output of the following code:

```
#include<stdio.h>
main()
{
int I, sum, num;
{
    clrscr();
    for(sum = num = i = 0; i<10; i= i+1)
    {
    printf("Enter a number");
            scanf("%d", & num);
                    if(num <= 0)
                            continue;
                    sum = sum + num;
            }
    printf( "The sum of positive numbers entered is %d\n" , sum);
    getch();
    return 0;
}
 }
```

Arrays

An array is a collection of variables of the same data type that are referenced by a common name. Multiple data items that have similar characteristics such as range of numeric values are placed into an array where they will all share the same name. Example: a1……a9. They must all be of the same data type. Such arrays store only numeric values. Each element of an array can be referred to by an array name and an index or subscript. You can also use arrays to store and manipulate characters using character arrays.

Syntax

```
<data_type> <variable_name> [<dimension_size>];
```

Example

int a[10], where the array name is a, and there are 10 data items all of the same integer / numeric data type. The array consists of 10 elements. The index value is 0. Each array element is referred to by specifying the array name followed by one or more subscripts, with each subscript enclosed in square brackets. The number of subscripts determines the dimensionality of the array. For example, *a[i]* refers to an element in the one-dimensional array a. Similarly *a[i][j]* refers to a two dimensional array *a*.

Defining Array

Arrays need to be defined similar to other data types along with the size of array (The number of elements in the array). An array can be of a single dimension, two dimension or can also be multi dimensional.

Example

One dimensional array

int a[10];

char str[30];

Two dimensional array

int a[5] [3];

A two dimensional array is declared with the number of rows and the number of elements in each row. In the above example there are five rows and each row has two elements.

Array Initialisation

Arrays can be initialised during the array declaration. While initializing arrays, the values should be given within curly braces and the values should be separated by commas. All the elements of the array should be of the same data type. An Array cannot be initialized with another Array. To copy all the elements of one array to another array, you have to copy each individual element to another array.

Example

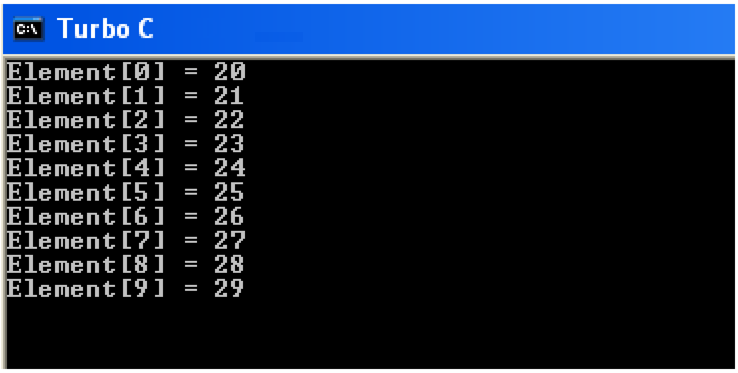
`int arr[5] = {10,20,30,40,50};`

Example

CODE

```
#include <stdio.h>
int main ()
{
   int a[ 10 ]; /* a is an array of 10 integers */
   int x,y;
    /* initialize elements of array a */
   for ( x = 0; x < 10; x++ )
   {
      a[ x ] = x + 20; /* set element at location x to x +
20 */
   }

   /* output each array element's value */
   for (y = 0; y < 10; y++ )
   {
      printf("Element[%d] = %d\n", y, a[y] );
   }
    return 0;
}
```

RESULT – Output Screen



```
Element[0] = 20
Element[1] = 21
Element[2] = 22
Element[3] = 23
Element[4] = 24
Element[5] = 25
Element[6] = 26
Element[7] = 27
Element[8] = 28
Element[9] = 29
```

Character Arrays

A variable of char data type can store only one character at a given point of time. But if you want to store a string of characters then Character Array is used. A character arrays has string of characters that terminates with a NULL character.

Syntax

```
<data_type><variable_name>[<dimension_size>];
```

Example

```
char name[30];
char month[15];
```

The first line defines an array variable called *name* of, character data type that can store a maximum of 30 characters. The second line defines an array variable called *month* of character data type that can store a maximum of 15 characters.

Array Initialisation

The character arrays can be initialised at the time of declaration in two ways as shown below:

```
char name[] = "Sandhya";
char month[4] = {'M','A','R',’\0’};
```

In the last statement, the array has been initialised using a string. Each character is assigned to the appropriate index automatically. The null character is appended automatically.

Note

If the null character is not specified when the array elements are initialised individually, as in the second example above, null will not be appended automatically. When null character is not specified the printf function reads the subsequent memory location and prints unrelated characters. To avoid improper output you have to explicitly append the null character at the end of the array.

Printing Character Arrays

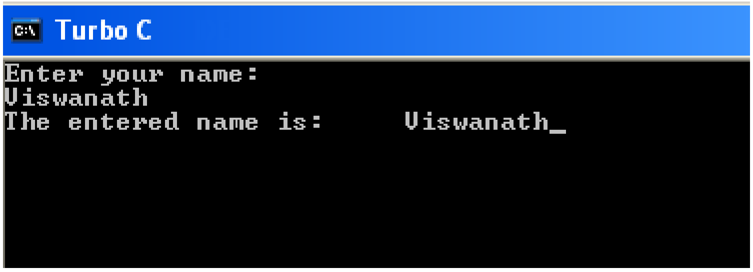
A character array can be printed by passing the name of the array to the printf() function. The printf() function displays all the characters in the array until the null character ‘\0’ is reached. The format code for printing a character array is “%s”.

Example

CODE

```
#include <stdio.h>
int main ()
{
   char name[ 10 ]; /* name is an array of characters that
                       can store name with 10 alphabets */
   printf("Enter your name: \n”);
   scanf(“%s”, name);
   printf(“The entered name is: \t %s”, name);}
```

RESULT – Output Screen



```
Enter your name:
Viswanath
The entered name is:     Viswanath_
```

EXERCISE

1. Write the output of the following code.

```
#include <stdio.h>
main()
{
char uname[4];
name[0] = ‘J’;
name[1] = ‘A’;
name[2] = ‘Y’;
printf(“The name of the student is %s”, uname);
}
```

2. Write a program to accept a string and find its length.
3. Write a program to accept 10 numbers using an array and find the largest and smallest of the numbers.
4. Write a program to accept marks of 10 students, print the entered marks and the sum of marks.
5. Write a program to accept marks and name of 10 students, print the name, marks, total marks and the average of marks.

Library Functions

The C library contains a collection of executable pieces of code that can be reused in a program. These predefined functions help you to perform specific tasks easily. These functions are called *Library Functions. The Library Functions can be either Built-in Functions or user defined functions that* are stored under header (.h) files. Library functions help in avoiding repetition in the program code.

The header files contain information that is required to make use of the C libraries. You can make use of the library functions by just including the appropriate header file in which the functions are defined. You must include these header files along with the pre-processor *#include* at the beginning of each program.

Mathematical Functions

All the mathematical functions are stored in math.h file. The following table lists and describes in brief about the various mathematical functions.

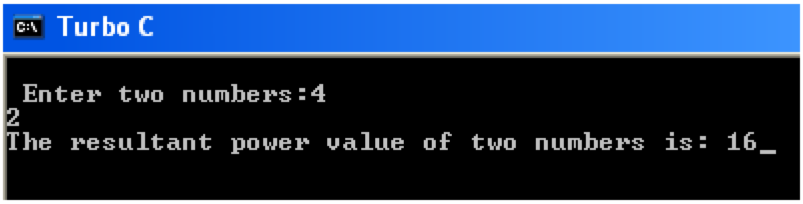
Function	Description
sqrt()	Calculate the square root of a number.
pow()	Calculate the value of a number raised to the power of some specific value.
sin()	Calculate the sine of the input value. It takes only one argument and it should be in radians.
cos()	Calculate the sine of the input value. It takes only one argument and it should be in radians.
tan()	Calculate the tangent of the input value. The input value for this function should be in radians.
log()	Calculate the logarithmic value of the input number. The number can be a whole number or floating number. The data type of the return value is double.
exp()	Calculate the exponential of a number. The input number can be a whole number or fractional number.

Example

CODE

```
/*Program to find the power of two numbers using power of
function*/
#include<stdio.h>
#include<math.h>
void main()
{
Int n1,n2;
long res;
clrscr();
printf(”\n Enter two numbers:“);
scanf(“%d %d”, &n1,&n2);
res=pow(n1,n2);
printf(”The resultant power value of two numbers is: %ld”,
res);
}
```

RESULT – Output Screen



```
 Enter two numbers:4
2
The resultant power value of two numbers is: 16_
```

Note

clrscr() in the above code clears the screen. The clear screen function is inbuilt with the stdio.h header file. The clrscr() – clear screen is a simple function used to clear the contents of the output screen. This function neither accepts any argument nor returns any value. If you are using an earlier version of C compiler then add the header file conio.h to include the clrscr() library function for clearing the screen. #include <conio.h>

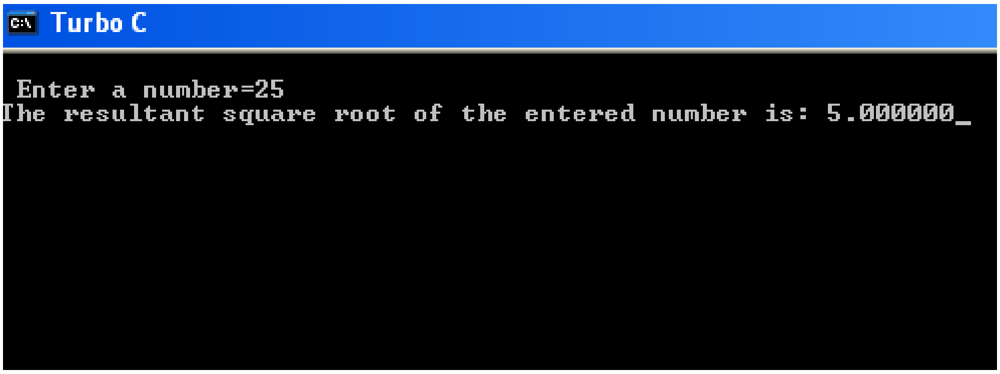
In the above example program the resultant variable is “res” that holds the resultant value. The output statement uses %ld type conversion as the variable is of long data type. Use a type version according to the type of variable declared in your program.

Example

CODE

```
/*Program to find the square root of a number using square
root function*/
#include<stdio.h>
#include<math.h>
void main()
{
int n1;
double res;
clrscr();
printf(”\n Enter a number:“);
scanf(“%d ”, &n1);
res=sqrt(n1);
printf(”The resultant square root of the entered number is:
%f”,  res);
}
```

RESULT – Output Screen



```
 Enter a number=25
The resultant square root of the entered number is: 5.000000_
```

In the example program shown above, the Variable that stores the input has been declared as floating point and the resultant variable that stores the calculated result has been declared as double. In the output statement the %f type conversion has been used to display the value stored in the variable of double data type, (res).

String Function

The string function has all the inbuilt string manipulation header files. All the string functions are included in the string.h header file. The following table lists and describes in brief about some of the string functions.

Function	Description
strcat()	Used to concatenate two strings. This function is passed with two arguments.
strlen()	Used to find the length of a string. It returns the number of characters in a string excluding the NULL character.
strcmp()	Used to compare two strings. This function returns a value 0 when two strings are equal. strcmp() function compares both the strings and returns a value which is the numeric difference between the ASCII values.
strcpy()	Used to copy a string from one variable to another.
strrev()	Used to reverse a given string.
strlwr()	Used to convert a string to lowercase.
strupr()	Used to convert a string to uppercase.

The gets and puts Function

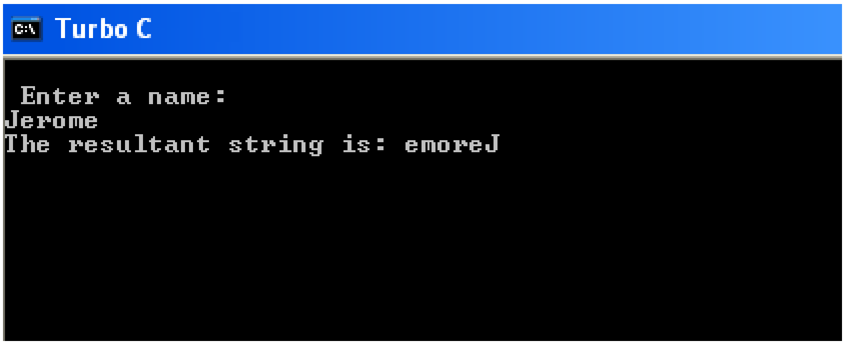
The `gets` and `puts` function facilitates the transfer of string from the computer keyboard to the C compiler. It performs the function of `scanf` and `printf` for reading and displaying strings.

Example

CODE

```
/*Program to find the reverse of a string */
#include <stdio.h>
#include <string.h>
main()
{
char s[15];
clrscr();
puts(“\n Enter a name”);
gets(s);
strrev(s);
printf(“ The resultant string is: %s”, s);
getch();
}
```

RESULT – Output Screen



```
 Enter a name:
Jerome
The resultant string is: emoreJ
```

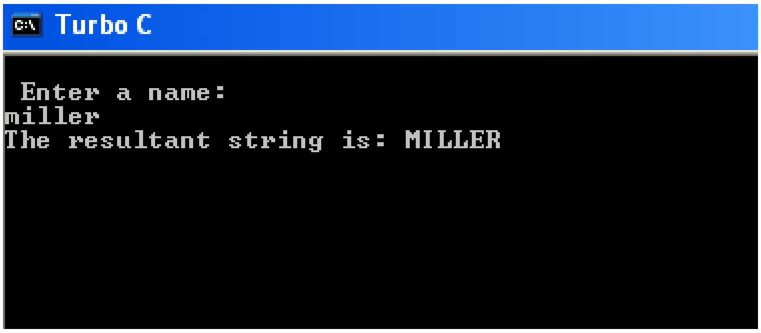
Note getch() in the above code is used to pause the command output screen to view the output.

Example

CODE

```
/*Program for converting a lower case string to upper case
string*/
#include <stdio.h>
#include <string.h>
main()
{
char s[15];
clrscr();
puts("\n Enter a name”);
gets(s);
strupr(s);
printf(" The resultant string is: %s”, s);
getch();
}
```

RESULT – Output Screen



```
 Enter a name:
miller
The resultant string is: MILLER
```

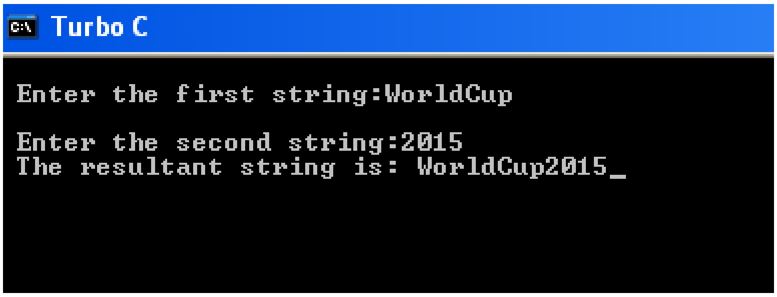
In the example program given above both the *gets()* and *puts()* function are used to get the input from the keyboard and output the value onto the screen. The *strupr()* function is used to convert a string to upper case character string.

Example

CODE

```
/*Program to illustrate concatenate two strings */
#include <stdio.h>
#include <string.h>
main()
{
char s1[50];
char s2[50];
char s3[50];
clrscr();
printf("\n Enter the first string:");
gets(s1);
printf("\n Enter the second string:");
gets(s2);
strcpy(s3,s1);/* Copies the string value of s1 */
strcat(s3, s2); /* Combines the second string with the first
                 string */
printf(" The resultant string is: %s", s3);
getch();
}
```

RESULT – Output Screen



```
Enter the first string:WorldCup

Enter the second string:2015
The resultant string is: WorldCup2015_
```

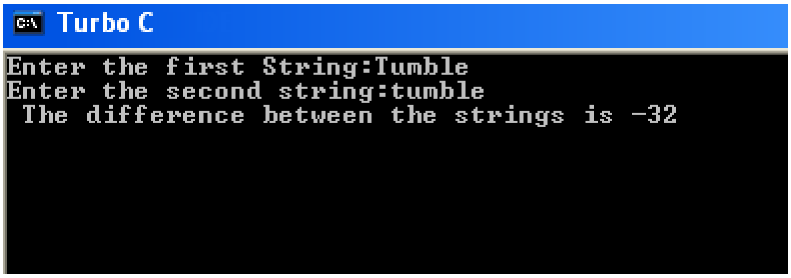
In the example program given above three arrays of character data type are declared to store 50 characters. Using the gets() function the string is entered from a keyboard. With a strcpy() function the s1 string is copied to s3. The strcat() function is used to combine s2 string with s3 string.

Example

CODE

```
/*Program to illustrate string comparison*/
#include <stdio.h>
#include <string.h>
main()
{
char s1[10];
char s2[10];
int diff;
clrscr();
printf("Enter the first String:");
gets(s1);
printf("Enter the second string:");
gets(s2);
diff = strcmp(s1, s2);
printf(" The difference between the strings is %d", diff);
getch();
}
```

RESULT – Output Screen



```
Enter the first String:Tumble
Enter the second string:tumble
 The difference between the strings is -32
```

In the above example the two array variables s1 & s2 are declared. The *gets()* function facilitates transfer of strings between the computer and the input device. The *strcmp()* function compares both the strings and returns a value -32 which is the numeric difference between ASCII values of T and t. The ASCII value of F is 70 and that of f is 102, which is 70 minus 102. Thus the ASCII difference '-32' between the first two non-equal characters is displayed.

EXERCISE

1. Write a program to accept name and convert it into upper case if it is in lower case.
2. Write a program to accept your name and display the same in the reverse order.
3. Write a program to check whether the string is a palindrome (Hint: Use strrev() & strcmp() functions).
4. Write a program to find the square root of the first ten natural numbers using the for loop.
5. Write a program using the while loop to find the logarithm value for the numbers entered by a user. Repeat the process until a user presses ‘N’.
6. Write a program to accept numbers and calculate the sine, cos, tan and pow of the numbers.